Calhoun
Institutional Archive of the Naval Postgraduate School

Calhoun: The NPS Institutional Archive

DSpace Repository

Theses and Dissertations | 1. Thesis and Dissertation Collection, all items

1999-06

Modterrain : a proposed standard for terrain representation in entity level simulation

Henderson, Dale L.

Monterey, California. Naval Postgraduate School

http://hdl.handle.net/10945/13514

Downloaded from NPS Archive: Calhoun

http://www.nps.edu/library

Calhoun is the Naval Postgraduate School's public access digital repository for research materials and institutional publications created by the NPS community. Calhoun is named for Professor of Mathematics Guy K. Calhoun, NPS's first appointed -- and published -- scholarly author.

Dudley Knox Library / Naval Postgraduate School
411 Dyer Road / 1 University Circle
Monterey, California USA 93943

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

MODTERRAIN: A PROPOSED STANDARD FOR TERRAIN REPRESENTATION IN ENTITY LEVEL SIMULATION

by

Dale L. Henderson

June 1999

Thesis Advisor:	Arnold H. Buss
Second Reader:	Leroy A. Jackson

Approved for public release; distribution is unlimited

DTIC QUALITY INSPECTED 4

19990915 064

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate o any other aspects of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 1999	Master's Thesis

4. TITLE AND SUBTITLE
MODTERRAIN - A PROPOSED STANDARD FOR TERRAIN REPRESENATION IN ENTITY LEVEL SIMULATION

5. FUNDING NUMBERS

6. AUTHOR(S)
Henderson, Dale L.

7. PERFROMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
US Army TRADOC Analysis Center
PO Box 8692
Monterey, CA 93943-0692

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this report are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12.a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release, distribution is unlimited

12.b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This thesis develops a standard Application Programmer's Interface (API) for modular terrain representation. The API hides the details of a terrain representation from an entity level simulation, thereby enhancing interoperability and flexibility. Additional contributions include reduced development costs, enhanced flexibility for developers, and the use of a component approach applicable to future simulations. Three reference implementations are developed in the thesis representing widely used terrain representations. These prototypes consist of a standard set of terrain services that can be used by a simulation developer without any knowledge of the underlying implementation. The prototypes serve as references, proof of the concept, and as tools for comparison and analysis of existing terrain algorithms. We demonstrate this comparison with the JANUS and Modular Semi-Automated Forces (MODSAF) line of sight algorithms. This set of API implementations also allows emerging simulations to use different terrain formats at run-time without source code changes. The API developed in this thesis is the basis for a United States Army Modeling and Simulation standard nomination.

14. SUBJECT TERMS	15. NUMBER OF PAGES
	85
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	18. SECURITY CLASSIFICATION OF ABSTRACT	19. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	

NSN 7540-01-280-550

Standard Form (Rev. 2-89)
Prescribed by ANSI Std. 239-18
298-2

Approved for public release; distribution is unlimited

MODTERRAIN: A PROPOSED STANDARD FOR TERRAIN REPRESENTATION IN ENTITY LEVEL SIMULATION

Dale L. Henderson
Captain, United States Army
B.S., United States Military Academy, 1989

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
June 1999

Author: [redacted]

Dale L. Henderson

Approved by: [redacted]

Arnold H. Buss, Thesis Advisor

[redacted]

Leroy A. Jackson, Second Reader

[redacted]

Richard E. Rosenthal, Chairman
Department of Operations Research

Abstract

This thesis develops a standard Application Programmer's Interface (API) for modular terrain representation. The API hides the details of a terrain representation from an entity level simulation, thereby enhancing interoperability and flexibility. Additional contributions include reduced development costs, enhanced flexibility for developers, and the use of a component approach applicable to future simulations. Three reference implementations are developed in the thesis representing widely used terrain representations. These prototypes consist of a standard set of terrain services that can be used by a simulation developer without any knowledge of the underlying implementation. The prototypes serve as references, proof of the concept, and as tools for comparison and analysis of existing terrain algorithms. We demonstrate this comparison with the JANUS and Modular Semi-Automated Forces (MODSAF) line of sight algorithms. This set of API implementations also allows emerging simulations to use different terrain formats at run-time without source code changes. The API developed in this thesis is the basis for a United States Army Modeling and Simulation standard nomination.

TABLE OF CONTENTS

I. INTRODUCTION 1

A. Purpose 1
B. Goal 2
C. Overview of Army Modeling and Simulation Standards 4
D. Challenges 5
E. Standards Nomination and Approval Process (SNAP) 7
F. Organization 8

II. TERRAIN REPRESENTATION 9

A. Elevation Models 10
B. Features Models 14
C. Terrain Model Usage 19

III. THE LINE OF SIGHT ALGORITHM 21

A. LOS Algorithm Components 22
B. Get Elevation Methods 23
C. Existing line of sight (LOS) algorithms 26

IV. THE APPLICATION PROGRAMMERS INTERFACE 33

A. Definitions 35
B. Data Types 36
C. Low Level Services 39
D. Meta Data Services 41
E. Basic High Level Services 42
F. Advanced High Level Services 44

V. REFERENCE IMPLEMENTATIONS 47

A. Terrain Representation Implementations 47
B. API Implementations 50
C. Example Experiment 50
D. Comparison of terrain sheets 52

VI. CONCLUSIONS AND RECOMMENDATIONS 55

A. Use of Prototypes 55
B. Nominated Terrain Standard 56
C. Web Based Simulation 56

TABLE OF CONTENTS

APPENDIX A. THE JANUS TERRAIN 59

A. JANUS Representation Scheme 59
B. Polygonal Features 60
C. Feature sub-types 62

APPENDIX B. MODTERRAIN SOURCE CODE 67

LIST OF FIGURES

Figure 1. Develop - Test - Develop Process 3
Figure 2. Elevation Posts 12
Figure 3. Elevation Grid Cells 12
Figure 4. One to One and Onto Mapping Between Feature Sets 17
Figure 5. Overloaded Mapping to a Feature Set 18
Figure 6. Mapping Using a Master Feature Set 18
Figure 7. The JANUS Get Elevation Method 24
Figure 8. The ModSAF Get Elevation Method 24
Figure 9. Nearest Post Get Elevation Method 25
Figure 10. The Southwest Corner Get Elevation Method 25
Figure 11. JANUS Line of Sight After Ref. [2] 27
Figure 12. DYNTACS LOS After Ref. [2] 28
Figure 13. ModSAF Line of Sight After Ref. [2] 30
Figure 14. Bresenham Line of Sight After Ref. [2] 31
Figure 15. Traditional Terrain Component Design 33
Figure 16. Simulation Design Using an API 34
Figure 17. API Diagram 35
Figure 18. Alternative Implementation of API 43
Figure 19. Terrain Sheet Generated Using ModSAF 52
Figure 20. Terrain Sheet Generated Using JANUS 52
Figure 21. Terrain Sheet Generated Using Nearest Post 53
Figure 22. Analytic Surface Sampled at Terrain Resolution 53
Figure 23. Analytic Surface Over Sampled at Four Times Terrain Resolution 54

Executive Summary

This thesis develops a standard Application Programmer's Interface (API) for modular terrain representation. The API hides the details of a terrain representation from an entity level simulation. Separating terrain interface from implementation in this manner enhances simulation interoperability, reduces development costs, and enhances flexibility for developers of Department of Defense simulation models. Development of the interface included examination of the existing body of simulation terrain components and the literature on simulating terrain.

In this research we found a small set of critical terrain services common to all of the existing terrain simulation techniques. The interface attempts to capture those functions that are common to a majority of these legacy simulations. The interface divides these functions into high and low level. The thesis documents these high level and low level services by defining method names, parameters, return types, and explanations. Low level services provide access to the underlying terrain data structure while high level services provide answers to model questions like line of sight and movement. As a default, these are written in terms of low level services, and the use of an API permits selective extension and replacement of specific functions without re-engineering the entire terrain component. The API data structure defines the default variable types used by compliant implementations.

The thesis includes three functioning prototype implementations of the API based on the geometric methods used by the JANUS, Modular Semi-Automated Forces, and CASTFOREM simulations. The API makes it possible to use identical terrain representa-

tions in comparing algorithms such as line of sight calculations. The thesis recommends further experimentation using the prototypes, use of the prototypes as components in a modular simulation architecture, and extension of the concept of components to other classes of simulation problems. The thesis is the basis for a United States Army Modeling and Simulation standards nomination.

ACKNOWLEDGEMENT

The author wishes to acknowledge the patience and support of Adrianna L. Henderson throughout the preparation of this thesis.

Just beyond the town there were two hills. One was wooded and green; the other was flat, topped by a cemetary. The Union commander, a tall blond sunburned man named John Buford, rode up the long slope to the top of the hill, into the cemetary. He stopped by a stone wall, looked down across the open flat ground, lovely clear field of fire...[1]

I. INTRODUCTION

Terrain has had a pivotal impact on the outcome of combat actions for all of recorded history. Consequently, military leaders have studied terrain since antiquity. In modern times researchers have attempted to capture those aspects of terrain that are critical to determining combat outcomes and to develop ways to simulate this impact.

Buford's decision to secure the ridge overlooking Gettysburg is only one of countless military decisions made throughout history in which terrain figured decisively. Had the same forces fought the battle of Gettysburg on different terrain, the outcome might have been far different. Hence there is some consensus among simulation developers that no entity level model of ground combat can be complete or properly balanced without representing the impact of terrain. The environment impacts nearly every aspect of combat [2].

A. PURPOSE

Existing and legacy entity level computer generated forces (CGF) simulations are complex and tightly coupled to their terrain components. This thesis develops a standard Application Programmer's Interface (API) for modular terrain representation. The API hides the details of a terrain representation from an entity level simulation. The standard modular terrain interface contributions include advances in simulation interoperability, reduced development costs, enhanced flexibility for developers, and the use of a component approach applicable to future simulations. In addition to the interface design, reference implementations are developed for this thesis. These prototypes consist of a standard set of terrain services that allow a simulation developer to use a set of common terrain services without any knowledge of the underlying terrain representation.

The API developed in this thesis is the basis for a United States Army modeling and simulation standard nomination. This standard nomination is the aim of the ModTerrain project that is part of the TRADOC Analysis Center Fiscal Year 1999 research plan. The thesis directly supports the United States Army Modeling and Simulation Office

(AMSO) terrain standards category by developing an API for terrain representation in CGF simulation [3].

B. GOAL

The ultimate thesis goal is to develop a software interface that contains a complete set of services for modeling terrain. Prototype implementations of this interface demonstrate that simulation developers may choose from a number of terrain models by simply writing calls through the API. In short, the goal is to abstract the terrain implementation details from the simulation and the simulation developer. The development phase of the API included a thorough review of the existing analytical simulation terrain representation methodologies. The current version of the API is the consequence of iteration on the early draft specifications presented to the Army M&S community for review in late 1998 and early 1999. These draft and subsequent revisions to the API were an attempt to break down those aspects of terrain, terrain services, and terrain simulation that were common to at least 90% of the simulations examined in the literature. Software, algorithms, and interfaces developed and prototyped in this thesis will be applicable to most existing simulations and useful to the entire M&S community.

In this thesis we:

- develop the first versions of the API - These versions of the API were disseminated to several Army M&S organizations for review. This iterative process of API design and review builds consensus for the final product.
- provide a foundation on which to base the standard - The API standard should be based on an examination of the literature and the large body of legacy simulations. From this examination we were able to distill the fairly small number of critical functions common to all entity simulations that use terrain. This produced a broadly applicable, yet compact specification in the end.
- document the research leading to the nomination - By coupling a thesis to the development of a standard nomination, we were able to capture the background and decision making processes that led to the overall design of the specification.
- prototype reference instances of the API for immediate study - The prototypes developed in this thesis are functioning implementations of the API. Their development proved that it is possible to abstract terrain. They exist now for further development, exploration, and experimentation.

- create possibilities for further research - The existing reference implementations are tools for simulation research, comparison of terrain methodologies, and demonstration. A major demonstration of the API is scheduled for mid 1999.

This process of develop - test - develop began in the fall of 1998 and is ongoing. It is described in Figure 1 below.

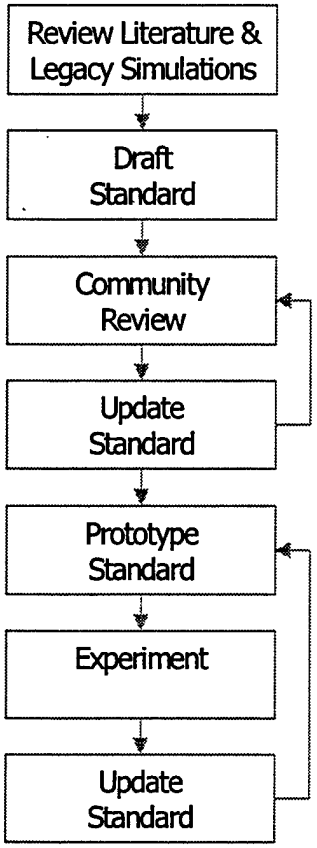

Figure 1. Develop - Test - Develop Process

This thesis is the foundation of the ModTerrain project whose goals are:

- to provide simulation composability,
- to support improved simulation interoperability,
- to reduce simulation development costs,
- to support other modeling and simulation standards,
- to provide flexibility for simulation developers,
- to allow for high performance implementation, and
- to foster innovation in future simulations [3].

C. OVERVIEW OF ARMY MODELING AND SIMULATION STANDARDS

The United States Army uses modeling and simulation in three distinct but related domains:

- Training, Exercises & Military Operations (TEMO),
- Research, Development & Acquisition (RDA), and
- Advanced Concepts & Requirements (ACR) [4].

The world of Army modeling and simulation encompasses a broad array of simulations that are built on an equally broad array of modeling methodologies. This host of methods, simulations, interfaces, representations, and models has grown over several decades of model development, reliance on model output, and the iterative and progressive validation of modeling methods. Three categories of simulation are used by DoD: live, constructive, and virtual.

Live simulations are those carried out by real soldiers operating real equipment firing either live engagements at targets or simulated engagements at each other. The best example of a live simulation is the National Training Center (NTC) in the Mojave Desert in California. Here leaders, staffs and soldiers participate in brigade and battalion level force on force exercises using real weapons fitted with the multiple integrated laser engagement system (MILES). MILES resolves all direct fire engagements. Indirect, air-to-ground, mine and other attrition events are also modeled in the exercise. As described in the Combat Training Center Handbook, "units, equipped with Weapons Engagement Simulation Systems, conduct training in areas containing sophisticated data collection and recording systems that provide a record of engagement for review, analysis, and use in planning and conducting training upon return to home station [5]."

Virtual simulations are high fidelity digital representations of combat presented to the soldier, leader, or staff in some form of synthetic or simulator environment. The most obvious example of a virtual simulation is the flight simulator. Flight simulators present a pilot or flight crew with a convincing representation of the outside world viewed from within a functional mock-up of their aircraft cockpit. Virtual simulation permits repetitive training of dangerous or expensive procedures. Examples include complex malfunctions,

emergency procedures or expensive missile gunnery. The virtual domain is almost exclusively used for training, although some advanced concepts testing and human factors experimentation use virtual simulations.

Constructive simulations are those that take input on the environment, systems, processes, and interactions of combat, use these inputs to generate outcomes, and provide the analyst with output based on these outcomes. Constructive simulations fall into two general classes: aggregated and entity-level.

An example of a constructive simulation used for training is the Battle Command Training Program (BCTP) Based at Fort Leavenworth, Kansas. Part of this program is a constructive simulation called a "War Fighter Exercise" (WFX). The exercise uses the Corps Battle Simulation (CBS). The WFX is conducted at the unit's home station.

> "Division, Brigade, and Corps Tactical Operations Centers (TOCs) deploy to field locations, normally within 15 km of the installation's BSC. Inside the BSC unit players simulate the subordinate units of the corps/division and fight the battle using the CBS. BSC players communicate with their higher headquarters using doctrinal means of communication only, making the simulation transparent to the commanders and staffs [5]."

D. CHALLENGES

Recent constrained budgets along with marked gains in computer technology have driven increased reliance on simulation in the live, virtual and constructive domains. The increased demand for simulation to fill an expanding set of roles led to a situation in the early 1990s in which overlapping, vertical, non-standardized development threatened to fragment efforts to develop timely, accurate, inexpensive, and useful simulations. As noted by McGlynn and Timian, "we examined the current state of the Army's M&S environment, or more simply put, where we were. Then we articulated the desired state, or where we wanted to be, in the form of an objective M&S environment. We then set about establishing a course of action to bridge the gap between the current state and the desired state [6]." This examination and analysis led to the development of the standards nomination and approval process (SNAP) and the Model and Simulation Resource Repository (MSRR). Critical to the success of this effort to create a process for modeling standards was its broadly defined objective "to create an environment that promotes the sharing and

reuse of M&S Standards procedures, practices, processes, techniques, algorithms, or heuristics [7]." In so defining the process, Army leaders and analysts avoided the formation of a rigid authoritative organization and maintained the leverage inherent in a system that eliminates the boundaries between the military, industrial, and academic development of models and simulations.

Within SNAP existing standards are classified as draft, approved, or mandatory and are maintained in these 19 different standards categories:

- Acquire,
- Architecture,
- Attrition,
- C4I Integration,
- Command Decision Modeling,
- Communication Systems,
- Cost Representation,
- Data,
- Deployment/Redeployment,
- Dynamic Atmospheric Environments,
- Functional Description of the Battlespace,
- Logistics,
- Mobilization/Demobilization,
- Move,
- Object Management,
- Semi-Automated Forces,
- Terrain,
- Visualization, and
- Validation Verification & Accrediation [7].

The ModTerrain project directly addresses standards requirements for the terrain, semi-automated forces (SAF), and object management categories.

The terrain standard category "establishes standards for the objects, algorithms, data, and techniques required to represent terrain and dynamic terrain processes in modeling and simulation [3]."

The semi-automated forces category "includes software integration that produces realistic entities in synthetic environments that interface appropriately with live, constructive and simulator entities, but which are generated, controlled and directed by computer routines [3]."

The object management category "is involved with the process that develops abstract object classes that are consistent in their representation of object attributes/methods, applicable to 95% of the M&S employing objects, understood by the M&S community, and interoperable at levels allowed by their model environment [3]."

E. STANDARDS NOMINATION AND APPROVAL PROCESS (SNAP)

The Army uses a seven step process, SNAP, to develop M&S standards [8]. These steps are:

- Build Teams,
- Define Requirements,
- Develop Standards,
- Achieve Consensus,
- Obtain Approval,
- Promulgate Standards, and
- Educate.

The current state of the ModTerrain project is the development of the standard. Prototyping and working models will assist in the development of consensus and the resolution of interoperability and other issues from the broader community. The ModTerrain API is unlikely to evolve into a rigid, mandatory standard applicable across the entire spectrum of M&S activities. Rather it is being deliberately developed as a flexible, open framework for enabling simulation developers to build on existing terrain representations and services.

F. ORGANIZATION

The remainder of this thesis contains a discussion of terrain representation, a description of the ModTerrain API standard, documentation of the reference implementation, and examples of some experimentation permitted by the reference implementation.

Chapter two describes some of the general methods of representing terrain elevation models and terrain features models. We examine different terrain representations for illustration and comparison with emphasis on explicit regular gridded terrain elevation representations. Finally we look at the general information a typical analytic simulation model requires of a terrain representation.

Chapter three describes in detail the structure of the most common analytic line of sight methods. The line of sight algorithm is broken down into constituent subroutines, and the DYNTACS, JANUS, ModSAF, and CASTFOREM line of sight methodologies are explained.

Chapter four is a complete description of the Application Programmer's Interface. It lists the object hierarchy and the low-level, high-level, and meta services specified by the API along with parameters, return-types, and brief descriptions.

Chapter five is a description of the reference implementation developed for this thesis. This reference implementation includes three instances of the API modeled on the geometric terrain services provided in the JANUS, ModSAF, and CASTFOREM simulations.

Chapter six is a series of recommendations for further research using the tools developed in this research. These include the continued use of the prototype for exploration, the incorporation of the prototype into component based simulations, and the extension of the process of component abstraction to other classes of simulation problems.

II. TERRAIN REPRESENTATION

All entity level simulations, even board games and rehearsal tools, make use of some kind of terrain model. At one end of the terrain model spectrum are "sand tables." These are crude scaled down mock-ups of an operation constructed from materials at hand. Commanders and staffs in small units use sand tables principally for mission rehearsal and war gaming because they help leaders to visualize terrain, spatial relationships and the progression of the tactical plan thorough time [2]. At the other end of this spectrum are high fidelity digital virtual representations. The Military Operations in Urban Terrain [MOUT] training facility is a terrain model of a general urban setting that might be used in a live simulation. Similarly, the National Training Center [NTC] is a general desert terrain model. Special operations forces occasionally use faithfully replicated building and airplane mock-ups for live training and mission rehearsal. In military board games a scale terrain board with an overprinted grid and relief data represents terrain. Terrain characteristics in these board game models are usually aggregated to the level of resolution of the grid cell.

A terrain representation is distinguished from raw terrain data by at least one level of processing and is stored so that a specific simulation can access it directly. A terrain representation is normally generated from raw data for use by the simulation or generated by some landform surface generating algorithm. The terrain representation is then stored in a format that can be accessed directly by the simulation. Such formats include pointers, headers, meta-data and an array structure of x coordinate, y coordinate, and z coordinate (elevation) values. The actual structure of these arrays varies by simulation. The interchange of data between different simulations is the subject of the Synthetic Environment Data Representation and Interchange Specification (SEDRIS). The goal of the SEDRIS project is to provide a common representation model and provide an interchange mechanism between terrain representations from different simulations [9].

A complete terrain representation requires consideration of both a terrain elevation model and a terrain features model. In the next sections we will examine the types of elevation and feature models common in existing simulations.

A. ELEVATION MODELS

In general, elevation data are stored as an array of numerical values that represent a uniform, discrete sample of the terrain surface. An individual element in this array is known as an elevation post. Most existing representations use regular, square grids of elevation posts. Others use regular rectangular, triangular or hexagonal representations. These representations are called regular gridded networks (RGN). The most general case of an explicit terrain representation uses irregularly spaced elevation posts that generates a terrain mesh of non-uniform triangular facets. These representations are known as Triangulated Irregular Networks (TIN). TINs may be generated from a regular gridded network by selectively removing elevation posts using thinning algorithm.

The underlying array of elevation posts is a discrete representation of the continuous real world. One may conceptualize progressively higher and higher resolution terrain representations, but at any level there are an infinite number of "holes" between every two posts.

The only polygons capable of mapping without overlap to a regular grid are uniformly sized squares, rectangles, triangles, or hexagons. Squares are used most commonly in digital representation [12]. The advantage of regular gridded networks is the simplicity of finding a post near or at a specific location. Given the extent and resolution of the grid, the calculation to find the element or elements nearest to a general point is trivial. The elements may be stored in a one dimensional array, and the explicit x and y coordinates may be either stored or calculated at run time. A general location of interest will always be bounded by a polygon whose vertices consist of points of known elevation. Once the bounding polygon is found these elements may be used to interpolate the elevation of the general point contained within the bounding polygon. In a TIN this method of pulling the bounding polygon from a node list is more complex, but the methods used for determining the elevation of the location in question are similarly trivial.

The elevation at the exact coordinates of an elevation post is stored explicitly in the terrain elevation database and may be easily obtained. However, the elevation of a location offset from a post must be approximated at run time. The need to repeatedly poll the terrain representation for elevations is a large computational burden. By some estimates the more complicated JANUS line of sight (LOS) algorithm is three times slower than a simple nearest post LOS algorithm [12]. The subroutine for determining elevation is an important component of this algorithm, and the implementation of a software method for determining the elevation of a general point within the boundaries of the terrain representation is critical to efficient, useful simulation.

Some simulations simply return the elevation of the nearest post to the point in question. Some perform more complex arithmetic to estimate an elevation. Those that return the elevation of the nearest post are sometimes called "grid cell" or "nearest square methods [12]." More complex algorithms, like those used by ModSAF and JANUS, return an interpolated value for the elevation. Elevation models fall into the broad categories of explicit, analytic, and hybrid. These are described in the next sections.

1. Explicit Representations

Any explicit representation of terrain must consist of some systematic arrangement of elevation data. The individual element in this set is called an "elevation post."

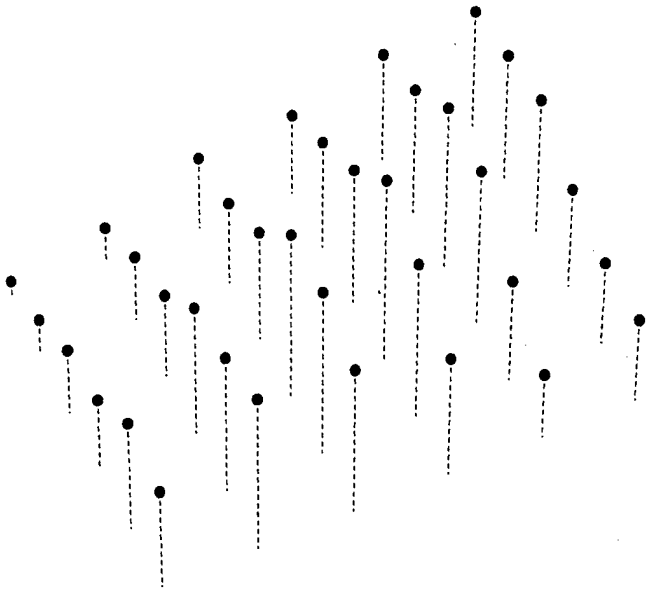

Figure 2. Elevation Posts

Alternatively, these posts may be viewed as grid "cells" of known elevation.

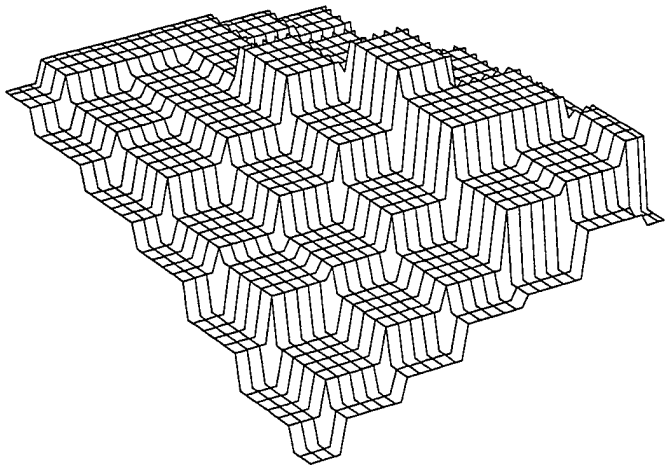

Figure 3. Elevation Grid Cells

Explicit representations may be geo-specific or geo-typical. Geo-specific terrain is developed by sampling elevations somewhere on the earth's surface. It is commonly used

when we wish to study a known area for which elevation data exist. Geo-typical terrain representations are usually built from algorithms that invoke Gaussian or trigonometric formulae to generate a smooth surface. These functions are generally parameterized so that they produce a landform with desired characteristics that are typical of some portions of the earth's surface, but specific to none.

2. Analytic Representations

Models that use analytic terrain representations use similar functions to represent the surface. However, an analytic representation invokes the representative surface generating function for an elevation value at run time instead of querying a pre-processed database of terrain posts. These methods generally alleviate storage requirements, add some computational burden, and permit "exact" rather than interpolative determination of the elevation of the terrain sheet at a specific point. They do not permit the incorporation of real world data. An example is the Variable Resolution Terrain Algorithm. This algorithm builds a terrain surface from the superposition of a number of hills whose parameters are determined stochastically within a range corresponding to the overall terrain characteristics desired [11].

3. Hybrid Representations

One may conceive of a representation in which known elevation posts are used as initial conditions for surface generating equations that can fill in the spaces between them. As an example, sand dunes tend to be found in relatively large fields of regularly spaced, similarly shaped dunes. Because of the low resolution in many explicit representations an individual dune would not likely be represented in a typical terrain elevation database. A hybrid representation can bring the dunes into existence for the associated combat model.

4. Comparison of Regular and Irregular Representations

The overriding disadvantage of regular terrain grids is that they are wasteful of storage space in cases where large areas of terrain are of nearly identical elevation. The general TIN representation is a more efficient method of depicting the actual contour.

However, the advantage of the regular network is that it is both easily generated from existing data that are generally stored in regular grids and that the logic for using a regular network is much simpler than the logic for using an irregular network. The DTED products are a uniform matrix of terrain elevation values that provide basic quantitative data for systems and applications that require terrain elevation, slope, and/or surface roughness information [12]. These data exist for the entire earth at various levels of resolution and general methods of storage and retrieval are easily implemented and translated. Translations between different representations, datums, and coordinate systems are easily implemented.

The elevation model provides only a limited view of the terrain. A complete terrain picture requires features. Feature representation is the subject of the next section.

B. FEATURES MODELS

The simplest terrain representation makes use only of elevation data. This is sometimes called a "bald earth" representation [11]. An array of x, y, and z coordinates is sufficient to specify the entire terrain mesh. However, entities interact with more than just the ground underneath them. The real world environment is filled with features that impact behaviors, interactions, and combat outcomes. These influences include breaking line of sight and impacting mobility. The terrain grid alone is likely too simple a model to provide credible results. Terrain representation must also permit the modeling of terrain features. The richness of this feature set depends on the simulation requirements. A driving force behind the feature data study cited above was the "need for a more precise understanding of the cultural and natural feature impacts on LOS prediction [12]." Features are often classified as point, linear, or area [9]. These classes are described below.

1. Point Features

Point features are natural or man-made objects, like buildings, towers, or individual trees, that may be adequately located in the terrain representation by a single point in space. The feature carries information about its physical extent and interaction behaviors as parameters. While the virtual representation must create an adequate sensory image

from these data carried by the feature, the constructive model need only carry feature parameters needed by the underlying simulation methodology. In a regular gridded network these point features may be placed accurately at general positions or offset from their real positions and placed coincident with elevation posts. Similarly a TIN may define a post at the location of each point feature or simply place the point features at general locations not associated with a vertex in the network.

2. Linear Features

Linear features include roads, rails, waterways, power lines, and similar terrain elements whose spatial properties may be adequately modeled as one-dimensional. Often these objects carry information like lane width, surface type, and water depth as parameters rather than as explicit geometric information. Linear features models, like point and area feature models, have a conflict between the virtual simulation's requirement for data to support visual models and the constructive simulation's requirement for data to support analytical models as point feature models.

3. Area Features

Area features contain information about a region with extent in more than one dimension. Forested areas, swamps, general urban terrain, and lakes are examples. These features may be defined by a list of posts, a list of coordinates defining a bounding polygon, or as an attribute of a defined region of the elevation model such as a triangular facet in a TIN.

4. Scope of the Feature Representation

The range or richness of the set of terrain features represented is dependent on the needs of the underlying simulation methodology. One major consequence of this varying need for feature data over the current family of simulations is a tight coupling between terrain representation and simulation. The variety of feature data used by simulations nearly rivals that found in commercial geographical information systems (GIS).

Feature representation is an area in which legacy, contemporary and proposed terrain representations diverge dramatically. Additionally, the needs for feature data vary widely by intended use of the terrain representation. For example, consider the representation of a building. A virtual visualization requires data like color, reflectiveness, luminance, and surface texture. In fact extremely high-resolution representations of complex features like buildings require a large and specifically defined data structure. In contrast, a building may be represented adequately in a constructive simulation by a short list of data elements, such as its corner locations, and height. There are many enumeration scheme standards and libraries for features data. An example is the Military Specification for Vector Smart Map [10]. This specification includes a nearly exhaustive list of terrain feature types and sub types. The focus of the specification is on cartographic products like maps and navigational charts, so the feature set is organized into "thematic layers" that are an reflective of the art of map making rather than the art of simulation. Each enumeration scheme tends to be highly representative of the needs of the team who built the representation. One of the goals of the SEDRIS project is a set of mappings between many of these feature enumerations.

Adding to the complexity of feature representation is the notion of an exhaustive enumeration of feature types. Legacy simulations attempt to capture a large enough set of feature attributes to allow for a detailed simulation methodology while remaining within speed and storage constraints. These restrictions on the scope of terrain attributes led to restrictions on the simulation methodologies that were dependent on the terrain. Movement modifiers in the JANUS representation are carried only for three broad classes: wheeled, tracked, and dismounted entities. This places a limit on any modification to the movement algorithms employed by the simulation. Any proposed enhancement requires altering the underlying terrain representation that renders the existing body of terrain representations obsolete. In the next section we examine the major ways in which simulations use terrain representations.

5. Mapping Between Representations

Mapping between feature representations in different terrain models is one of the most difficult problems facing the API design. An exhaustive set of one to many and complementary many to one mappings between terrain representations is not necessary because some simulations are designed to operate alone. Mapping rules may be devised which permit a useful interaction of any two specific simulations even if some loss, rounding, or generalization is necessary in one or both directions. Ideally, this mapping between feature sets is mathematically "one to one and onto" as shown in Figure 4. For each element of the first set there is exactly one, and only one, corresponding element in the second set. Typically these one to one and onto mappings are not possible. In fact among legacy systems there are no two simulations that use identical feature sets, so running the simulations together, each using its own feature set, permits the situation shown in Figure 5.

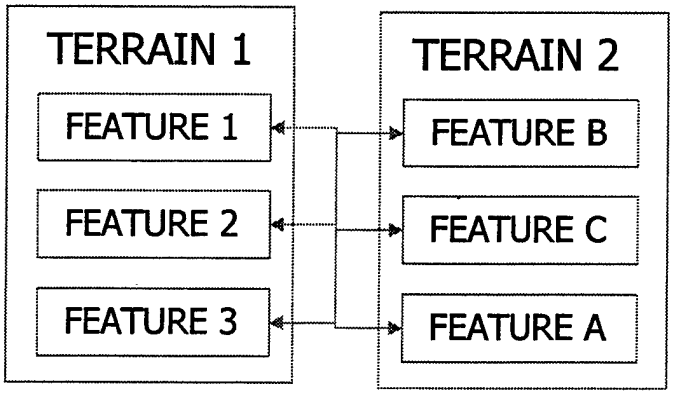

Figure 4. One to One and Onto Mapping Between Feature Sets

In Figure 5, the mapping to one feature in the set is overloaded. Note that the arrow from feature 2a to feature B is one-way. This implies that once a feature of type "2a" has

been mapped to a feature of type "B" in the other representation, it's identity as a "2a" feature can never be recovered deterministically.

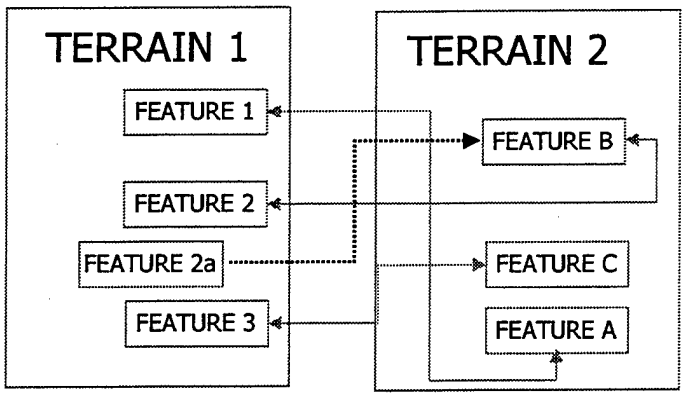

Figure 5. Overloaded Mapping to a Feature Set

Even if a master feature super-set, encompassing every conceivable feature type and sub-type in every simulation in existence were built, and even if all simulations were required to map into this feature set, the problem would exist. Consider Figure 6.

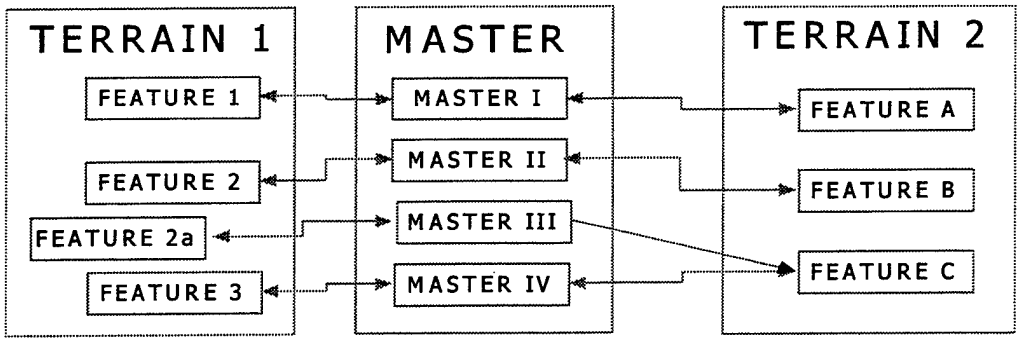

Figure 6. Mapping Using a Master Feature Set

In Figure 6 the master feature set is placed between the two run time terrain representations, but mapping from Terrain 2 back to Terrain 1 still mislabels all instances of feature type "2a" in Terrain 1 as feature type "2." The insight gained from this sort of analysis is valuable because it limits the scope of the API. In the prototypes developed in the thesis, the feature object is kept abstract enough to permit its instantiation as a representation of nearly any feature from an existing or future feature set. At the same time this feature object was designed to be both "thin" and "flexible." Thin refers to the small number of instance variables and methods associated with the feature object. Flexibility is a conse-

quence of the object oriented approach to the prototype implementation. This approach permits the extension of the feature object through inheritance.

C. TERRAIN MODEL USAGE

The degree to which the simulation uses terrain drives the development of the representation. One of the major stated goals of this thesis is to show that the tight linkage between terrain and simulation may be de-coupled through an abstracting interface. The places where terrain and simulation interact most frequently are described below.

1. Intervisibility, Detection, Acquisition

An entity level simulation will interact with its terrain representation most frequently to resolve intervisibility, acquisition and detection outcomes. In fact these services are a tremendous user of computational resources and together serve to constrain the resolution and manner of terrain use. Further, the entity by entity intervisibility polling requirement has driven most of these simulations to employ a time step or hybrid time step and discrete event methodology. A standard set of terrain services and a compliant terrain representation must permit efficient access to and resolution of these questions of geometry with respect to the terrain grid. Further, since "bald-earth" representations are not likely to be adequate, the impact of feature data on these questions must also be represented in an efficient manner.

2. Movement

After intervisibility, movement is the next most demanding consumer of terrain information. The physics of movement and the impact of terrain on movement are well understood and modeled in high resolution engineering simulations. Relevant data on terrain material, soil type, moisture, tree spacing and diameter are available. Their impact on movement is well modeled. As a result many simulations model the impact of terrain and weather on the movement of an individual entity with a great deal of resolution. Any standard set of terrain services must provide efficient access to the type of information these algorithms require.

3. Visualization

Although user interfaces are not simulations, visualization of the terrain model and the interaction between entities in a simulation is critical to extracting useful information from the simulation. A standard set of terrain services and a compliant representation will permit visual representation of the underlying analytical model. If the underlying model is a highly simplified version of the terrain, the visualization of it should not imply a more detailed representation. On the other hand if the purpose of visualization is to enhance realism or training effect, then the visualization software should be able to construct and populate a realistic scene from a sparse analytical terrain representation.

4. Other Uses

A well designed modular terrain component permits increased complexity in a modular way in areas where such complexity is needed. Aerosols, weather, and dynamic terrain elements, are all areas left open to enhancement by the design of the API. This openness is a critical design element of the API. Within the API the implementer is free to use existing functions or write others.

The most algorithmically intensive question, and one that has been studied extensively, is the question of line of sight. This is the subject of the next chapter.

III. THE LINE OF SIGHT ALGORITHM

This chapter describes the components of the geometric line of sight algorithm. It gives a detailed description of some well known line of sight methods which are the basis for the prototype implementations of the API. First the underlying methods of determining the elevation of an arbitrary point are described, then the line of sight algorithms that use this elevation value are exposed. This chapter touches on four established line of sight methods: Dynamic Tactical Simulation (DYNTACS), Modular Semi-Automated Forces (ModSAF), JANUS, and CASTFOREM.

If an entity level simulation requires interaction between entities, and if that simulation represents the effect of terrain, then it requires some method of determining whether an unbroken geometric line of sight between any two given entities or locations exists. This algorithm serves as a basis for determining whether a detection occurs or a direct fire engagement is possible. Further, this algorithm must posses certain characteristics to be useful in practical application. Among these characteristics are accurate output, repeatable results, computational simplicity, modest storage and retrieval overhead, and ease of implementation.

Accurate output refers to the need for the simulation to produce results that reflect sufficiently identical outcomes to those that would be encountered by identical systems operating on the real terrain modeled in the database. Repeatable results are easy to obtain in strict geometric models, but the notion may be extended to obtaining like or similar results in a statistical sense from different runs in a stochastic analytical model. The need for computational simplicity has driven most implementations of line of sight algorithms to trade accuracy for speed.

Computational complexity for the line of sight problem is polynomial in the number of entities, or n-squared; however, computational experience has shown that the line of sight algorithm is often the single largest consumer of processor resources in an entity simulation [15]. The extensive use of heuristics to reduce the number of required calls to

the line of sight algorithm leaves a problem that is still fundamentally polynomial and too computationally expensive for a modest number of entities.

Modest storage and retrieval overhead is related to computation efficiency, but is focused on the way the terrain model is represented, stored and retrieved. Increasing the resolution of the underlying terrain database is also an n-squared problem. A twenty-kilometer square regular grid of one-hundred meter elevation posts contains forty thousand elevation posts. Increasing that resolution to ten-meter intervals requires four million posts. In the case of irregular elevation grids, rapid database queries to provide elevation data become critical to the speed of the line of sight algorithm.

Ease of implementation is desirable because it allows simple prototyping, layered complexity and meaningful documentation. Further, ease of implementation enhances of verification.

The remainder of this chapter will consider line of sight algorithm components, several different methods for determining elevation, and finally, some established line of sight algorithms which use these elevation methods.

A. LOS ALGORITHM COMPONENTS

Consider a simulation of a battle between a force of x entities and a force of y entities. Let n be the sum of x and y, the total number of entities simulated. To a large extent the events in the simulation will be driven by detections, which will be predicated on line of sight. This requires a poll of each entity for its current line of sight status to every other entity at each time step or event in the simulation. As the number of entities increases, this problem grows with the square of the number of entities. Hence, for practical purposes simulations must make use of some filters and heuristics to decrease the number of instances in which the full LOS calculation must be invoked. In other words, the simulation developer must reduce, by some reasonable means, the number of entities that require resolution of the line of sight question for a given situation or time step. Otherwise, the simulation becomes overburdened with solving the line of sight question between every sensor-target pair. The simplest heuristic is to assume a maximum detection range and eliminate sensor-target pairs whose range exceeds this maximum.

The algorithm also needs a method for returning the elevation of an arbitrary point in the terrain database. The general structure of algorithms is to sample a set of points along the sensor target line and determine if any one point has a higher elevation than the actual sensor target line. Some algorithms sample only the nearest elevation posts to the sensor-target line. Others compare slope instead of elevation [15]. In a well-designed algorithm the function will halt and return "false" at the first indication that line of sight has been broken.

The final step in determining sensor-target LOS is determining whether features interposed between sensor and target prevent line of sight. Theoretically a sufficiently detailed feature representation allows geometric calculation of line of sight through the feature, but processor and storage issues have led most legacy simulations to use probabilistic methods. If geometric line of sight exists, a probability of line of sight is determined based on the number, type, and extent of features along the sensor-target line. We next examine several methods to get the elevation of a point.

B. GET ELEVATION METHODS

1. Four Point Linear Interpolation

In of the JANUS terrain methodology, the bounding polygon (four posts) of the queried position is used to estimate its elevation. Since four points do not define a plane, some method must be found to determine the elevation of a general position bounded by four known elevations. JANUS performs a linear interpolation on the four points [15]. Each of these points is an easily calculated distance from the position in question, and each has a known height. More accurately, the distance between the point in question and a projection of each of the bounding points onto a plane may be easily calculated. While this method produces a reasonable approximation of the elevation at a general point it imposes discontinuities in the terrain surface at the edges of grid cells. Since the original

JANUS terrain elevation resolution was in discrete units (pentameters), these discontinuities do not have a serious impact on the simulation.

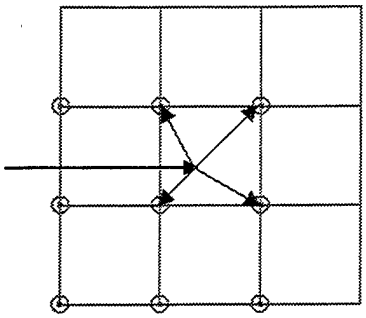

Figure 7. The JANUS Get Elevation Method

2. The SE to NE Assumption Method

The ModSAF simulation uses a similar terrain grid and line of sight algorithm to JANUS, but it does not estimate elevation in the same way. Instead ModSAF assumes a line drawn from the south-east to north-west corner of each cell. This produces a regular network of triangles. Instead of performing a least squares estimate for elevation the simulation determines onto which triangle the point in question projects. It then uses simple vector arithmetic to return an elevation [15].

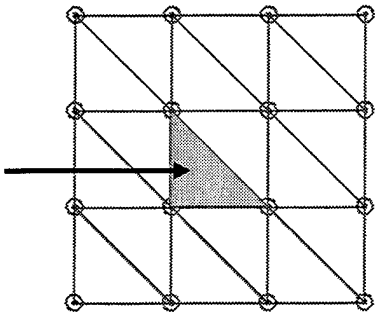

Figure 8. The ModSAF Get Elevation Method

This approach is also applicable for use in an triangulated irregular network where the facet surface is taken as the terrain surface. The method of determining the bounding polygon list will differ significantly, however, between a regular and irregular representation.

3. The Nearest Post Method

Some simulations approximate elevation by using the nearest post to the point in question. This method supports the Bresenham line of sight algorithm [12]. The great advantage of this approach is the speed with which the elevation is estimated.

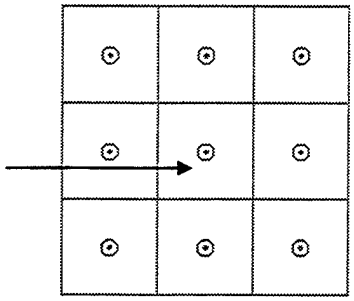

Figure 9. Nearest Post Get Elevation Method

This approach is identical to aggregating elevation data over a square "cell" defined by the resolution of the model. A similar method is used to aggregate elevation data over hexagonal arrays in some older simulations and board games [2].

4. The Southwest Corner Method

A simplification of the nearest post method is to return the elevation of the post in the southwest corner of the bounding square.

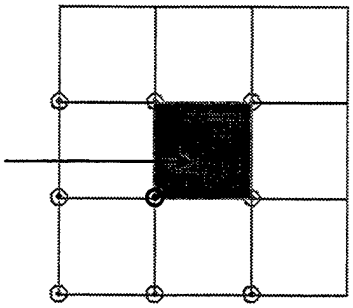

Figure 10. The Southwest Corner Get Elevation Method

This method offers the greatest speed advantage, but when used in a line of sight algorithm will not necessarily return a reciprocal result. Line of sight may exist in one direction and not in the other between two entities. [12].

C. EXISTING LINE OF SIGHT (LOS) ALGORITHMS.

1. General

The algorithms described here are only a representative subset of those that have been developed for use in constructive simulations. They are well described by Champion [12, 15] as well as Hartman, Perry, and Caldwell [14] and the Military Operations Research Analyst's Handbook [2]. The selected descriptions are used in the reference implementation of the API presented with this work.

2. JANUS LOS

We will examine the JANUS line of sight algorithm first. The JANUS LOS methodology strikes a reasonable balance between speed, fidelity, and simplicity. However, the JANUS LOS algorithm is very closely coupled with the JANUS terrain representation. Even though the algorithm and the representation are optimized for performance in the JANUS simulation, LOS still takes a significant percentage of the available processor time in a JANUS run.

This discussion of JANUS LOS will take three parts. First we must understand the JANUS terrain. Second we make use of the JANUS elevation interpolation methodology presented above. Finally, both of these discussions are used to build a description of the JANUS LOS algorithm.

JANUS uses a regular (square) gridded network of elevation posts in a select set of resolutions. Certain features within a JANUS terrain carry a parameter, PLOS, that impacts the line of sight calculation. Probability of line of sight (PLOS) is a value from [0,1] that is used in calculation to represent the probability that line of sight will exist along a 25 meter path through this feature. Features that carry this parameter include vegetation and urban areas. Once geometric line of sight using only the terrain grid had been confirmed, these probabilities are multiplied together to give a probability that line of sight will exist during a specific time step in the simulation.

To determine the elevation of a specific point in the terrain grid JANUS uses the four-point linear interpolation method described in the previous section. In order to carry

out the JANUS LOS algorithm, this four point linear interpolation must be executed several times.:

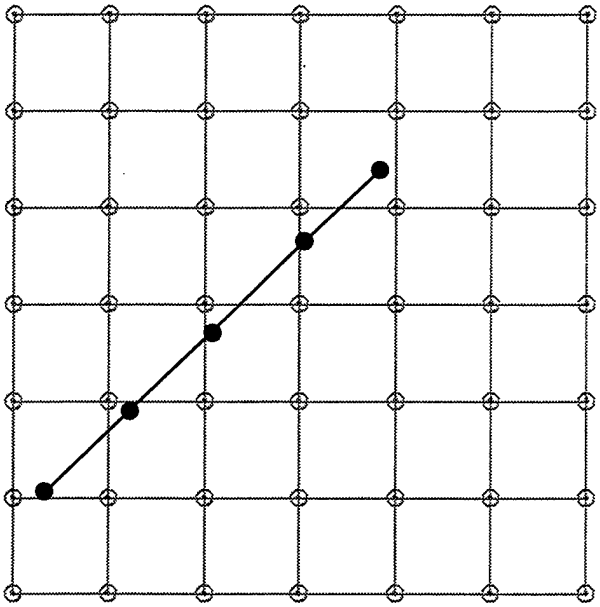

Figure 11. JANUS Line of Sight After Ref. [2]

Finally, in order to determine the probability that line of sight exists along a sensor target (s-t) line for one time step in a JANUS simulation run, the JANUS LOS algorithm goes through the following steps

For each iteration:

Calculate the X offset between sensor and target.
Calculate the Y offset between sensor and target.
Determine the largest offset.
Divide this offset value by the resolution of the terrain
Round to the nearest integer. (This is the number of evenly spaced discrete tests along the sensor target line that the algorithm will make).
Divide the s-t line into this number of equal length segments.
Determine the slope of the s-t line.
Step through each of these points along the s-t line one at a time

At each step:

Determine the elevation of the terrain grid at that point by four-point linear interpolation.
Compare this elevation with the elevation of the s-t line at this point.

If the elevation of the point exceeds the elevation of the s-t line at the point break and return "false."
If the elevation of the point is less than the elevation of the s-t line at the point then step to the next point.

Once all points have been checked, if geometric line of sight exists:

Determine if the path crosses any line of sight affecting features.
Determine the geometric distance traveled through any features along the path.
Divide these distances by 25 meters.
Determine the PLOS for each of these features.
Multiply through the PLOS times distance for each feature.

Return the result, a numerical value from [0,1].

Once the simulation obtains this value, it makes a draw from the uniform [0,1] distribution and returns true if that draw exceeds the computed PLOS. The existence or absence of line of sight then impacts many other functions of the simulation such as acquisition, firing, and visualization routines.

3. Dynamic Tactical Simulation (DYNTACS) Line of Sight

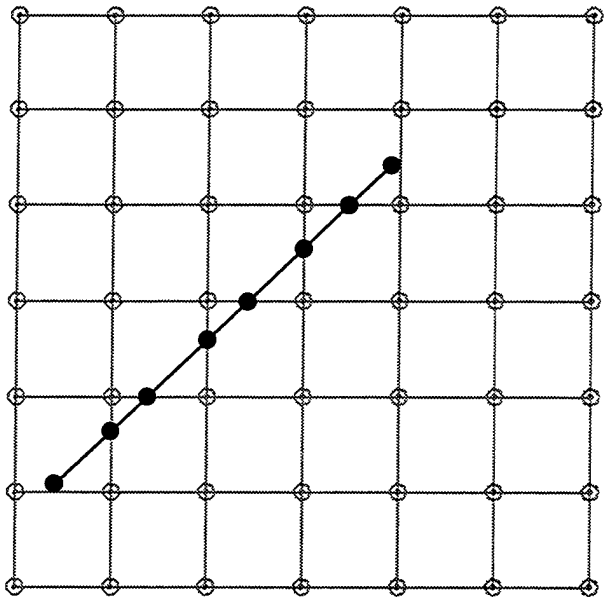

Figure 12. DYNTACS LOS After Ref. [2]

DYNTACS LOS uses a method similar to JANUS. The DYNTACS terrain representation is also a square grid of elevation posts. An identical four-point linear interpolation scheme is used to determine sensor and target elevation. Instead of breaking the s-t line into equal parts, DYNTACS LOS determines every point at which the s-t line crosses a facet edge in the square lattice. The algorithm then determines these elevations by linear interpolation on the two known elevation posts. These interpolations are deferred in code until their values are needed. DYNTACS steps through these grid crossing points from sensor to target and compares the sensor to target slope with the sensor to intermediate point slope. If the slope from sensor to intermediate point ever exceeds the s-t slope, the algorithm breaks out of the loop and returns false. If the algorithm steps through to the target it returns true.

4. Modular Semi Automated Forces (ModSAF) Line of Sight

ModSAF also stores terrain as a regular (square) gridded network of elevation posts. However, ModSAF terrain has a modified view of these elevation posts. This change allows for a smoother terrain sheet surface and eliminates the requirement for four-point linear interpolation as in JANUS LOS. ModSAF LOS assumes a diagonal across each square in the lattice of elevation posts. The orientation of this diagonal is northwest

to southeast. This diagonal and the resulting two triangles approximate the actual terrain surface.

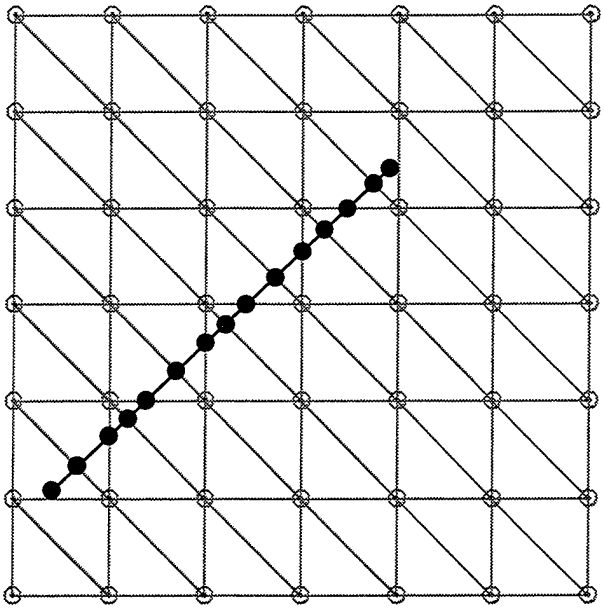

Figure 13. ModSAF Line of Sight After Ref. [2]

The LOS algorithm works in a manner similar to DYNTACS LOS except that the algorithm considers all of the s-t line intersection points including the diagonals. In addition, the elevations for sensor and target are determined from their coordinate and the three known points of the triangular facet on which they rest. ModSAF uses the same slope comparison technique as DYNTACS to determine whether LOS is broken along the s-t line. In general, the ModSAF calculation will run slower for a given terrain resolution than either the DYNTACS or JANUS representations [12].

5. Bresenham Line of Sight

As noted, the Bresenham method uses the elevation of the nearest post to the sensor and target, and the elevation of any grid cell through which the sensor-target line

passes. This eliminates the need for all of the interpolation arithmetic used in the JANUS, ModSAF, and DYNTACS LOS methods.

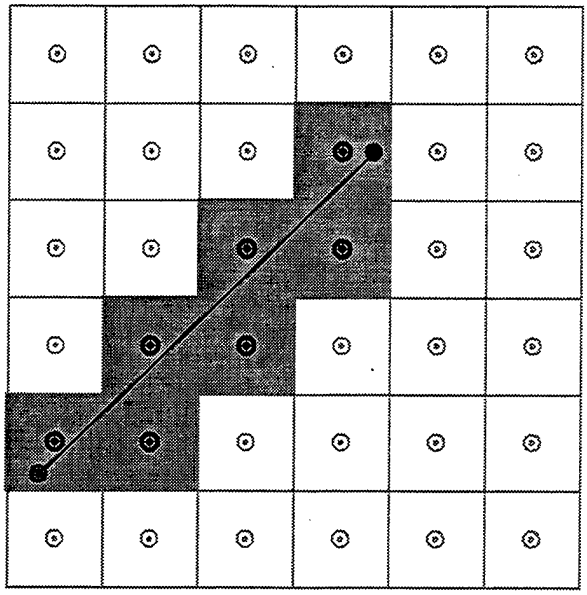

Figure 14. Bresenham Line of Sight After Ref. [2]

The method is extremely fast, but suspect when used in a low resolution terrain model [11]. The algorithm steps through the elevation cells and compares the elevation of the cell to an elevation threshold calculated from the slope of the sensor target line. In order to ensure reciprocal LOS consistency care must be take to "assign" cells consistently when a point in question lies an equal distance from two cell posts.

6. ALBE Implementation OF Bresenham

The AirLand Battle Experiment (ALBE) simulation used a similar algorithm to Bresenham line of sight except that it used the southwest corner elevation post. The general Bresenham method uses the "post centered grid cell" approach. The disadvantage of the ALBE implementation is that it permits non-reciprocal resolution of line of sight calculations [11]. This means that far a given sensor-target pair, the existence of line of sight from sensor to target is not sufficient to guarantee line of sight in the other direction. This lack of reciprocal agreement is a consequence of the method used to determine step size.

The step size can cause different elevation posts to be selected when stepping from sensor to target than from target to sensor [11].

7. Other Geometric Approaches

The methods described above are representative of a large class of existing geometric approaches to determining LOS. The implementations described in the next chapter are algorithmically faithful to these. They differ principally in their use of double precision real values rather than integers. Some older representations used integer values and integer arithmetic because of both computational speed advantages and the ease of mapping integer values to finite "pixel" displays. The API permits use of any algorithm.

The preceding chapters form the background neccesary for the discussion in the of the application programmers interface. Development of the API standard began after this background was thoroughly examined in a search for the principal themes, structure, and uses of terrain in simulation. The next chapter describes the API in detail.

IV. THE APPLICATION PROGRAMMERS INTERFACE

This chapter describes the API. First we show the differences between traditional design and design using an API. Next we show block diagrams of the default and alternative internal structure of the API. Finally, we step through the definitions, data types, and function calls required by the specification. This version of the API was used to write the reference implementations developed in this thesis.

In traditional simulation design, the terrain component is tightly coupled to the simulation. As illustrated in Figure 15, the simulation translates data about the environment into a run time representation that may be directly queried by the simulation during execution.

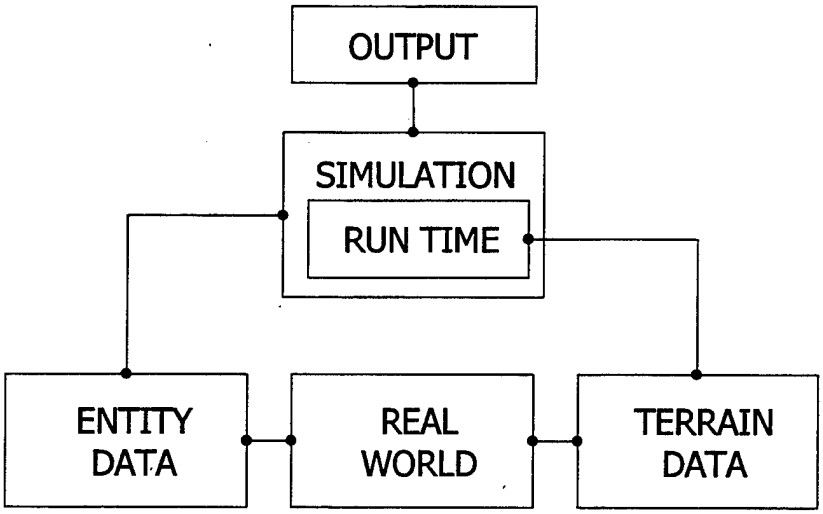

Figure 15. Traditional Terrain Component Design

By using the ModTerrain API to abstract terrain from the simulation as shown in Figure 16, the developer is able to de-couple the terrain representation. This makes compliant terrain representations interchangeable.

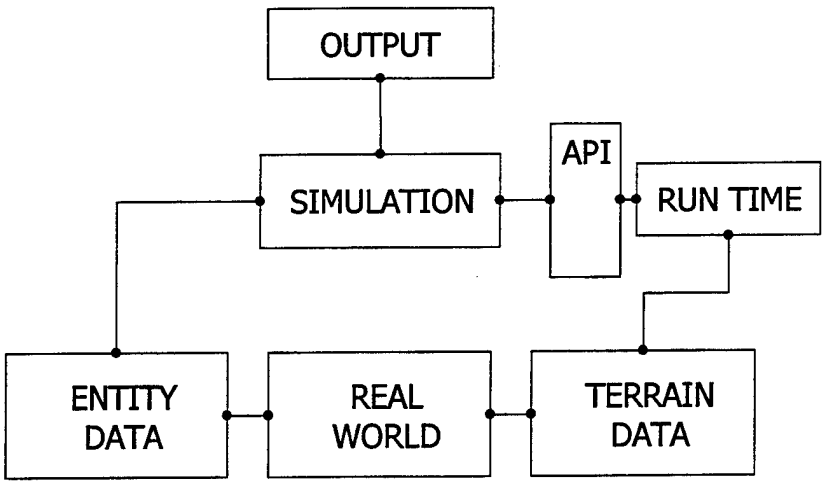

Figure 16. Simulation Design Using an API

The interface is a multi-level interface consisting of high and low level services. Low level services are those that access the terrain representation directly while high level services may be written in terms of low level services. The API provides the simulation developer with a completely specified set of terrain accessing services that hide or abstract from the simulation the details of the underlying terrain representation. As shown in Figure 17, the low level and meta services are the only services that must be implemented with direct access to the underlying terrain representation. This permits code re-use by allowing developers to implement alternative high level services strictly in terms of the existing low level services with no need to change any reference to the basic terrain representation.

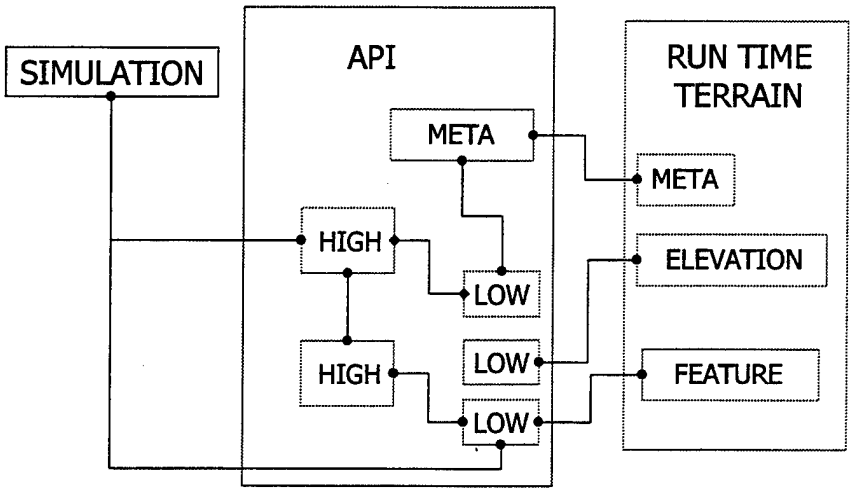

Figure 17. API Diagram

A. DEFINITIONS

In developing the API the development team attempted to use precise definitions in the specification and to extend these definitions to descriptive names in well defined data types. These data types may be implemented directly, or used to wrap existing data types via pass through functions. The formal definitions are spelled out in the API functional description document. In abbreviated form they are:

- position - the area or point occupied by a physical object.
- location - a position or site occupied or available for occupancy or marked by some distinguishing feature.
- coordinate - any set of numbers used in specifying the location of a point on a line, on a surface, or in space.
- distance - the degree or amount of separation between two points, lines surfaces or objects.
- direction - the line or course on which something is moving or is aimed or along which something is pointing.
- elevation - the height to which something is elevated or the height above the level of the sea.
- altitude - the vertical elevation of an object above a surface of a planet or natural satellite.

- height - the distance from the bottom to the top of something standing upright or the extent of elevation above a level.

B. DATA TYPES

Data types reflect a compromise between compatibility with legacy systems and openness. All of the data types specified in the API have direct correlation to similar data types in existing simulation models and terrain representations. Mapping from the defined specification data types to those used by a reasonably exhaustive range of simulations is trivial. The implementer need only write a pass-through that can wrap his data in the API compliant data types.

1. Mapping Between Data Types

Two way mappings without loss between externally defined data types may not be possible in all cases. One example is a mapping from floating point value to integer value and back. This can not be accomplished without loss of information, although it can be accomplished with a reasonably arbitrary degree of precision. Likewise mapping between supported feature sets in different representations without loss is not possible. A visual terrain representation that supports dozens of different tree types to permit a realistic display can be mapped easily to an analytic representation that only requires one tree type. But mapping that one tree type back to the dozens supported in the visual representation is impossible.

2. Coordinate

In this section we describe the API specification for a one dimensional coordinate. We specify a 64-bit floating point value as the default standard for coordinates in compliant terrain representations. We further specify the default unit of measure for coordinate as the meter. The mapping arguments above are still relevant. The use of floating point values to represent the basic unit of spatial measurement is a significant design choice that will require consensus. There are powerful numerical arguments for specifying integer values and using a fine grain as default.

In fact the development of euclidean geometric schemes, libraries, and techniques for use in computers has favored integer values because of two major advantages: the speed of integer arithmetic, and the integer values uses in discrete graphics display. We ultimately rejected specifying that coordinates be integer values with a default scale for representation in centimeters. Setting aside questions of computational efficiency, the most important consequence of constraining the representation of coordinate to integer values is the possibility of eliminating all numerical ambiguity in computation, storage, and translation from the representation. An important step in the standards nomination and approval process, however, is building consensus. The current consensus among those who have reviewed the ModTerrain draft standard is that the API specification should use floating point values. In the end the flexibility of floating point values outweigh numerical efficiency and implementation costs.

3. Location

A two dimensional location is a pair of coordinates. Within the API, two dimensional location is given the name *location_type*. Three dimensional location is given the name *location3d_type* and the third dimension is of *elevation_type* rather than *coordinate_type*. Higher dimension coordinate schemes are possible, but neither mandatory nor supported by the existing services in the API. We also specify *location_list* and *location3d_list* types that are generally defined data structure containers holding linked lists of their respective data types.

4. Distance

We specify a *distance_type* as a 64-bit float. This is a positive scalar value.

5. Direction

We specify a *direction_type* We specify a default scale for direction of radians clockwise from "representational north." We use a 64-bit floating point value to represent direction. Since the range of possible direction values is only two times pi it appears that this allows an unnecessarily fine representation of angles. The data type was selected for

consistency with coordinate and elevation and because there are conceivable high resolution requirements for very fine representation of angular measure. An example requirement is the accurate depiction of star fields in a rendered sky scene.

6. Elevation

The *elevation_type* is also a 64-bit floating point value. Elevation is distinguished from coordinate constructively by being defined in a separate data type because the notion of elevation is fundamentally different from the notion of coordinate. Specifying a location in three dimensions requires three coordinates. Knowing the elevation of the terrain at a point requires the two coordinates that specify that point, and the terrain elevation at that point.

7. Modifier

We specify a data type called *modifier_type* in order to permit methods that use or return data from the constrained range [0,1]. Modifiers can represent probabilities, percentages, or efficiencies. The modifier data type permits the terrain to carry simple concise information about its general impact on mobility, line of sight, degree of damage, and similar uses.

8. Enumeration

We specify an enumeration data type. This is an integer value that can be used to enumerate any general set of possible cases. The "enumeration value" to "enumeration meaning" mapping is not specified in the API. This makes the *enumeration_type* an extremely flexible tool within the API.

In the next section we describe the low level services that use these data types.

C. LOW LEVEL SERVICES

1. General

Low level services are those that reach directly into the terrain representation and return data in a form that is usable to the simulation at run time. The low level services must be tightly closely coupled to the representation. A fundamental goal of the API design was to limit and abstract the number of low level services required and supported. By doing this we greatly simplify the task of generating, altering, and mapping between compliant terrain representations.

2. Open Terrain

This service returns a *file_type*. It is the primary means the simulation will use to open the file containing a terrain representation for access by the API.

3. Close Terrain

In many languages efficient file and memory management require that files and databases be explicitly "closed." Pointers to their data are vacated and memory allocated to storage and retrieval from within them are returned to the system for general use. Such services will be provided here in an API compliant implementation.

4. Get Elevation

The get elevation method returns a value for the elevation of the underlying terrain representation at an arbitrary two dimensional point. As noted this is a critical function. In general the get elevation method must be written specifically to one or a most a small set of alternative data representations.

5. Get Nearest Elevation Post

The get nearest elevation post method is similarly tied to the underlying representation. It returns the nearest explicitly stored elevation value to a general point in question.

6. Get Elevation Post List

This method returns a list of elevation posts. The input parameters define a region on the elevation surface. These parameters consist of a location list and a distance. It has three required behaviors.

- A query given a single location and a positive distance returns those posts contained within a circle centered at the location and of radius given by the distance.
- A query given two locations and a positive distance returns those posts contained within a rectangle whose length is the distance between the two points and whose width is twice the distance value given.
- A query given n > 2 locations and a distance >= 0 returns those posts contained within the bounded area defined by those parameters.

7. Get Bounding Vertices List

This function returns the list of elevation posts that form the vertices of a polygon that bounds a general point in question. For example in regular square grid this function will generally return the four vertices of the square that bounds a point in question.

8. Get Feature List

This is the only function that permits the retrieval of data containing feature types. Its input parameters are a list or location and a distance as above, and also a list of feature types of interest. It returns the list of feature instances that are of a specific feature type and located by the same geometry rules used in the Get Elevation Post List method. Feature type and feature instance are highly abstracted under the API. This function places no requirements on either of these data types beyond the requirement that they be explicitly located in the coordinate system that underlies the terrain representation.

Implementing these low level services often requires access not only to the terrain data, but to information "about the terrain data." Access to this meta data is provided through services described below.

D. META DATA SERVICES

1. Get Elevation Model Type

This service is narrowly defined by the API, and is fairly restrictive; however, the impact on most implementations is minimal. The function returns an enumeration type whose value indicates the type of elevation model underlying the representation according to a standard enumeration scheme:

Value	Model Type
0	Unknown
1	regular gridded
2	irregular gridded
3	regular triangulated
4	triangulated irregular network
5	analytical

Table 1. Elevation Model Enumeration Scheme

2. Get Elevation Resolution

This meta data service returns a distance type whose value represents the minimum possible difference in elevation between any two posts. The API provides for varying resolution models by including a version of this function that takes location as an argument and return resolution in the vicinity of that location. For many applications these will return identical values. The developer is free within the API to define the vicinity of the point in question.

3. Get Horizontal Resolution

The API specifies similar functionality to provide a distance type whose value is a measure of the horizontal resolution of the model. The notion of varying resolution is similarly supported. In irregular networks the API requires return of an average resolution, but places no restriction on the determination of this value.

4. Get Terrain Boundary

This function gives the boundaries of the underlying terrain representation. Simulation developers are free to define appropriate entity behaviors while they are operating in an area that does not overly the terrain representation, but the API allows the simulation to know the boundaries of this representation. In the default this function returns a list containing two location types, the lower left and upper right corners. This list may be a series of vertices of an irregular polygon in a more complex implementation. Likewise a developer may, for some reason return boundaries well within or well outside the actual extent of included data. The contract between the terrain implementation and the simulation is that data exist within these bounds.

5. Get X Offset & Get Y Offset

The get X and Y Offset functions facilitate mapping and translation requirements. These may be simple translations from some physical two dimensional coordinate system or they may be composed of highly complex Spherical trigonometric functions to permit explicit mapping between planar and spherical surface models. The vast majority of ground entity level simulation is developed for examination of areas in which the plane approximation is valid. The API permits support of those instances where the model developer requires a more complex surface.

From the low level and meta services provided above, a default implementation of the API will construct the basic and advanced high level services described in the next two sections. The basic services provide the simulation answers to common geometry questions. The advanced services answer complex questions about movement and line of sight.

E. BASIC HIGH LEVEL SERVICES

1. General

In the reference, or default implementation of the ModTerrain API, the basic high level services are built from low level services. They provide the developer with basic information about spatial relationship and orientation. Developers are not bound by the

default implementations. Alternative implementations using user defined high level services and default or user defined low level services are all possible.

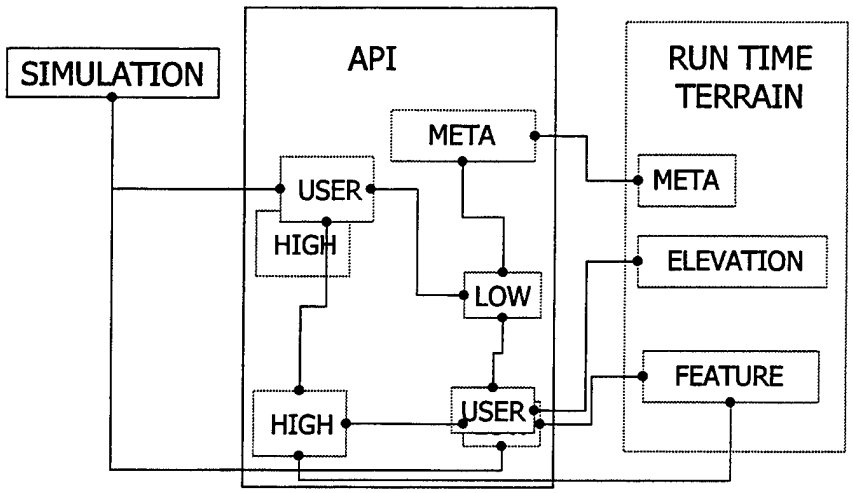

Figure 18. Alternative Implementation of API

2. Get Distance

This returns the distance between any two points. The specification gives a default implementation of euclidean two dimensional distance, but more complex implementations are permitted.

3. Get Direction

This returns the horizontal direction in radians subtended by a segment between two specified points and a line from the first point in the direction of a representational north. The API gives a default of "grid North" as the base line. Most computer representations of angle have a default of "screen right," or "grid West" as base angle.

4. Get Slant Range

Slant range is the line of sight distance between two different three dimensional locations.

5. Get Slope

Within the API slope refers to the instantaneous or average slope of the terrain representation. When this function is called with two location arguments, the slope between those two points is returned. When it is called with one location type and one direction type argument the instantaneous slope of the terrain representation at that location and in that direction is returned.

6. Get Translated Location

This function permits the simple translation of a location along a distance and in a direction. It returns a location type.

7. Get Material Type

This function return an enumeration type of the surface material present at the location given as an argument. Many movement algorithms account for varying surface material types in order to model their impact on trafficability. The underlying terrain representation must carry data about material type in order for the implementation to have use.

F. ADVANCED HIGH LEVEL SERVICES

The advanced high level services are built from basic services, meta services, and low level services and provide the developer with default implementations that may be used at a higher level in the overall simulation methodology. The advanced high level services are those that are most likely to be specific to an individual simulation, and are more likely to be overwritten by a simulation designer.

8. Line of Sight

This is the boolean value specifying whether unmodified geometric line of sight exists between two locations.

9. Probability Line of Sight (PLOS)

This is the probability [0,1] that line of sight exists between two locations. It permits developers to use stochastic methods of determining line of sight for detection and similar simulation functionality.

10. Movement Modifier

Similar to PLOS, this returns a value from [0,1] that can be used in a general way to modify the movement characteristics of an entity.

The next chapter describes the reference implementations developed by the author as prototypes to test the API and as tools for further study. The chapter goes on to describe some experiments developed to demonstrate the contrasting behaviors of the different implementations.

V. REFERENCE IMPLEMENTATIONS

We now present three reference implementations of the ModTerrain API. The run time terrain representation uses a regular gridded network for elevation geometry and a simplified feature representation scheme. The reference implementation all use the same feature services, and implement the geometric algorithms of the JANUS, ModSAF, and CASTFOREM simulations. The reference is written in the JAVA programming language. These implementations were specifically designed to support future work on the loosely coupled components and web based simulation projects.

A. TERRAIN REPRESENTATION IMPLEMENTATIONS

As noted, legacy simulations are closely coupled with their terrain representations. As a method of studying these representations, facilitating reuse at the code level, developing hybrid representations, and eventually paving the way for Loosely Coupled Representation of Terrain, we implemented the run time terrain object in a JAVA hashtable object. The hashtable is a data structure that consists exclusively of key-object pairs. The data types used as the keys and the data types that store the information in the hashtable are completely general. Access to these data is guaranteed through simple access methods that follow a standard pattern. This permits the use of a small set of key types for storing terrain information whose details may be unknown to the user. This run-time representation may be easily extended. These reference implementations and terrain data represent a resource for further study and development.

Key String	Object Description
"header"	a ModTerrain header_type
""xValues"	An array of x values of the posts
"yValues"	An array of x values of the posts
'elevValues"	A two dimensional array of elevation posts

Table 2. Run Time Representation Hashtable

Key String	Object Description
"horizontalResolution"	A double value giving the distance between posts
"elevationResolution"	A double value giving the resolution
"xCellWidth"	An integer giving the number of cells (E W)
"yCellWidth"	An integer giving the number of cells (N S)
"supportedFeatureList"	A vector of feature_type
"existingFeatureList"	A vector of feature_instance_type
"elevationModelType"	An integer giving the model type

Table 2. Run Time Representation Hashtable

1. The Terrain Class Hierarchy

As noted, a terrain representation consists of elevation data, feature data, and meta data. A ModTerrainData object contains variables and data structures to hold all of these data types along with methods that provide the ModTerrain API implementation access to these data. This object also includes a default (no parameter) constructor, a string constructor, and a constructor that takes an existing ModTerrainData object as its argument.

The class files that were written specifically to implement the ModTerrain data structure model are:

Class	Description
coordinate_type	stores/provides access to one coordinate in a double value
date_type	stores/provides access to a date as an integer value
datum_type	stores/provides access to the datum as an integer value
direction_type	stores/provides access to a direction as a double value [0, PI]
distance_type	stores/provides access to a distance as a double value
elevation_type	stores/provides access to an elevation as a double value
enumeration_type	stores/provides access to the enumeration as an integer value
feature_instance_list_type	stores/provides access to a list of feature instances
feature_instance_type	stores/provides access to an individual instance of a feature

Table 3. Data Structure Implementing Classes

Class	Description
feature_list_type	stores/provides access to a list of feature types
feature_type	stores/provides access to an individual type of feature
file_type	stores/provides access to a file as a String object
header_type	stores/provides access to a terrain representation header
height_type	stores/provides access to height as a double value
location_list_type	stores/provides access to a list of (coord, coord) locations
location_type	stores/provides access to a single location as (coord, coord)
location3d_list_type	stores/provides access to a list of 3d coord elev) locations
location3d_type	stores/provides access to a single 3d location as (coord, coord, elev)
modifier_type	stores/provides access to a modifier as a double value [0,1]
name_type	stores/provides access to a name type as a String object
vector3d_type	stores/provides access to a 3d vector as three double values
version_type	stores/provides access to a version type as an integer value

Table 3. Data Structure Implementing Classes

2. Terrain Grid Implementations

The terrain grid is implemented as a set of arrays. Time constraints have limited implementation only for regular gridded elevation representations. The extension to triangulated irregular networks demands the implementation of efficient sorting routines that can rapidly determine the specific triangular facet that contains an arbitrary point. It further requires a partitioning scheme that prohibits simultaneous assignment of an arbitrary point to more than one facet. The high level services developed in here could be used or adapted for use with an alternative terrain representation.

3. Terrain Feature Implementations

Each terrain feature is a specific instance of a feature from the set of features supported by the representation feature enumeration. Examples of *feature_type* and *feature_instance_type* are provided in the reference implementation.

B. API IMPLEMENTATIONS

We created three reference implementations of the ModTerrain API. These implementations are faithful to the geometric algorithms used in the JANUS, ModSAF, and CASTFOREM simulations respectively. The key numerical difference between these references and the original simulations is that they all use the same run time representation hashtable object whose numerical values are stored as double precision real numbers. In their legacy code these simulations used lower precision numerical schemes because of storage and computational efficiency constraints. They all use an identical simplified feature methodology that is patterned after the JANUS simulation. Each feature instance carries a [0,1] value that may be used as in JANUS to calculate the probability that line of sight exists across a path that contains the feature. Similarly each feature carries a [0,1] modifier value that may be used to represent the impact that this feature has on mobility. All of the implementations share common functionality for file handling and meta data services. This is an example of the degree to which implementation of an API permits code re-use. The following sections give an example of one use for these reference implementations.

C. EXAMPLE EXPERIMENT

As an example of the type of research that the reference implementations permit, we performed a simple experiment using instances of the JANUS and MODSAF implementations. The objective of the experiment was to determine if the line of sight algorithm used effected the probability of line of sight in a simulation. We generated a bald earth run time terrain representation from digital terrain elevation data, i.e. the data included no features. The terrain was a square approximately 8,300 meters on each side. For each trial we drew random sensor and target locations. For each of the sensor target pairs we tested for geometric line of sight using the JANUS and ModSAF line of sight algorithms. We performed 3000 trials to provide sufficient data for reasonable use of large sample approximations to the normal [16]. The null hypothesis under this experimental design was, "the

ModSAF and JANUS algorithms will return true for line of sight with the same probability under these circumstances" or more formally,

$$p_1 - p_2 = 0$$

We use the test statistic:

$$z = \frac{\hat{p}_1 - \hat{p}_2}{\sqrt{\hat{p}\hat{q} \cdot \left(\frac{1}{m} + \frac{1}{n}\right)}}$$

where:

$$\hat{p} = \frac{X+Y}{m+n} = \frac{m}{m+n} \cdot \hat{p}_1 + \frac{n}{m+n} \cdot \hat{p}_2$$

This large sample approach is taken from Devore [16]. The data are shown below in Table 4. Substituting these data into the equations above gives a value for the test statistic of -2.34, and under the conditions tested, we are able reject the null hypothesis at the $\alpha = 0.05$ level. This simple experiment confirms that the two algorithms generate out-

	JANUS	MODSAF
Number Of Trials	3000	3000
Number Line of Sight = True	564	495
Sample proportion	.179	.151

Table 4. Data from Multiple Line of Sight Tests

comes that are different enough to be significant. Although the magnitude of this difference is slight, the impact on a simulation that may calculate line of sight many thousands of times per run would almost certainly be measurable as a "hotter" rate of detection. While previous experimentation in this area was confounded by differences in underlying terrain representation, each of these algorithms ran on numerically identical terrain. This small experiment illustrates the type of analysis the prototype implementations permit. In the next section we describe a more qualitative experiment undertaken to give insight into the way different terrain methods "see" the elevation data.

D. COMPARISON OF TERRAIN SHEETS

We explored other experimental possibilities. One was the qualitative visual inspection of terrain sheets generated from the elevation methods of the three implementations. We generated a run time terrain that consisted only of elevation data. These data were themselves generated from the Digital Terrain Elevation Data (DTED) Level 1 data set for the southeastern United States. We then used each of the implementations to sample from these data at identical points offset from the explicit elevation posts. The sampling rate was roughly four to one. We rendered the generated sheets using the three dimensional chart generator provided with the S-Plus software package.

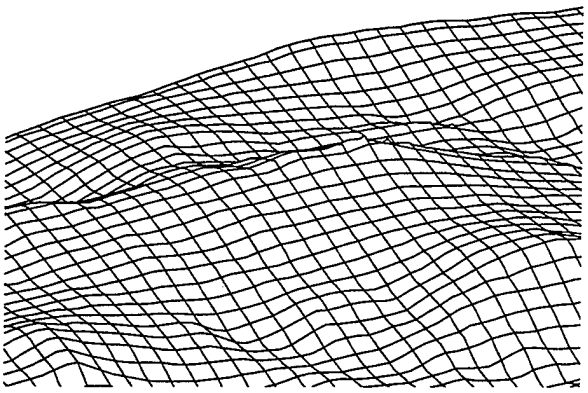

Figure 19. Terrain Sheet Generated Using ModSAF

Figure 19 above shows the general faithfulness of the ModSAF view of the terrain DTED level 1 data. The representation at least gives a visual perception of real terrain.

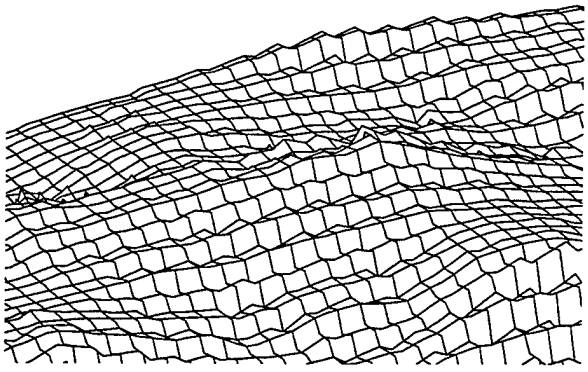

Figure 20. Terrain Sheet Generated Using JANUS

Figures 20 and 21 show views of the identical data as seen by the JANUS and CASTFOREM elevation algorithms. These share the ModSAF faithfulness to the DTED data, but at least appear blocky and less "real" to the observer.

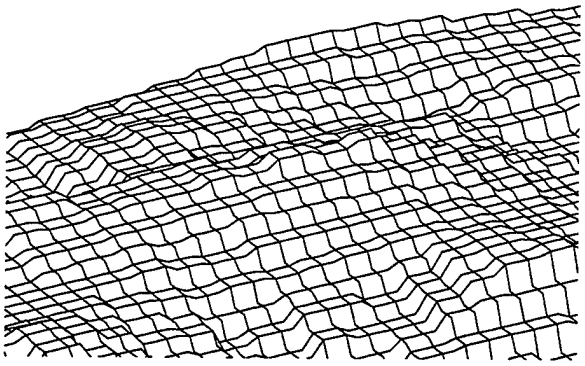

Figure 21. Terrain Sheet Generated Using Nearest Post

Clearly each of the algorithms view identical terrain data differently enough to be apparent in a casual visual inspection. We extended these visual assessments of "model views" of the terrain to consider a terrain generated from a mixed trigonometric and exponential level surface. In this analysis we examined the appearance of surfaces generated by sampling the surface near its level of resolution. We compared this to the appearance of a surface created by oversampling the data. Allow the scale of the existing terrain data to be a square 10,000 meters on a side. Let this square be a 101 by 101 element array of elevation posts spaced evenly at 100 meter intervals north south and east west. If we sample a section of the terrain surface at close to 100 meter intervals we note a somewhat different view of the terrain than if we sample it at 25 meter intervals.

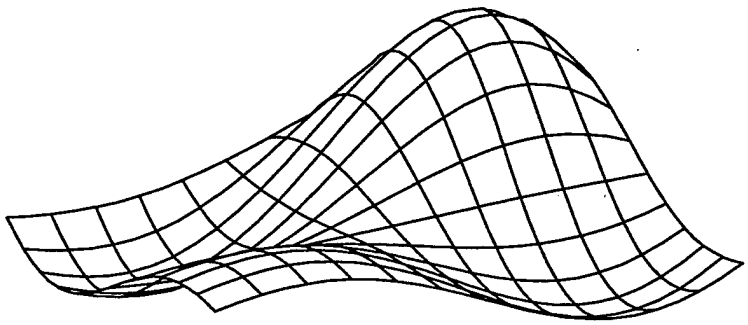

Figure 22. Analytic Surface Sampled at Terrain Resolution

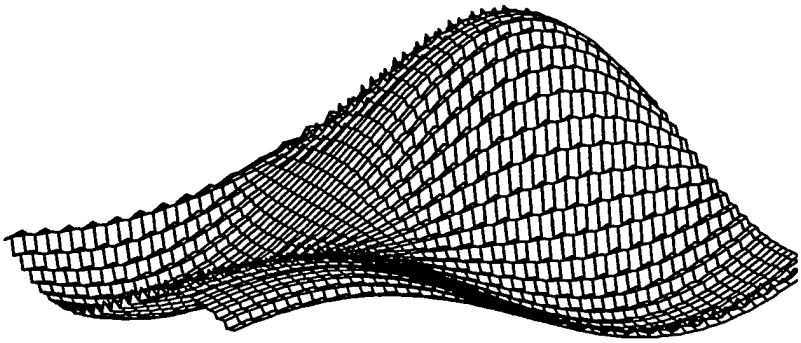

Figure 23. Analytic Surface Over Sampled at Four Times Terrain Resolution

The apparent difference in these views suggests experimentation on the impact of variable elevation resolution on simulation performance. This and many other related experiments are made possible by these reference implementations. The recommendations made in the next chapter include suggestions for experimentation with the prototypes.

VI. CONCLUSIONS AND RECOMMENDATIONS

This thesis provides a strong foundation for the ModTerrain project. It documents the research, iterative design process, and prototyping that have taken the program to its current state. The API is ready for full scale test as it moves toward release as an approved modeling and simulation standard.

The stand alone reference implementations permit analysis of different algorithms and representations independent of existing simulations. Legacy code wraps the terrain algorithms up in full simulations, while the prototype implementations permit controlled examination.

Rapid prototyping, evaluation, and side by side comparison similar to the example experiments shown in the thesis are all possible using the reference implementation and the tools available in Simkit. The JAVA programming language permits extension of the API using a large library of existing, efficient, verified code. The recommendations given here fall into three broad categories: recommendations for using the prototype implementations, recommendations for the API specification, and recommendations for extending the components notion to other classes of simulation functionality through the employment of similar APIs.

A. USE OF PROTOTYPES

The prototype implementations developed for the thesis may be used in two ways. First, they permit comparisons between the most widely used and understood terrain services using identical terrain representations. The design of these experiments is greatly simplified by the elimination of variations in performance due to numerical or machine differences. One obvious type of experiment compares the speed of various algorithm. Another possible experiment is a comparison of various algorithm results to measurements on actual terrain as an extension to Champion's work [15]. It is also possible to develop hybrid methods such as one that uses the ModSAF elevation model, but follows the JANUS feature model and line of sight algorithm.

Further, the implementations are available to serve as terrain components for simulation objects. In this way, researchers and students may explicitly use terrain in an entity simulation without a detailed study and time consuming re-creation of this work.

Most importantly, the prototype implementations may be used to explore methods of answering the line of sight question in a pure event step model. The algorithmically simple Bresenham instance is an appropriate first choice for use in these explorations, but the API structure permits its replacement with more complex algorithms as means are developed to use terrain explicitly in event step combat modeling. The prototypes developed here are not the only instances of the API under development as part of the ModTerrain project, however.

B. NOMINATED TERRAIN STANDARD

The ModTerrain API has been nominated to the Army Modeling and Simulation Office as a standard interface for abstracting the terrain component from simulations. The project includes experimentation on at least two implementations of the API. This experimentation will demonstrate that the principles demonstrated in this thesis can be scaled for use in fielded, commercial simulations. Other standard nomination teams will benefit from the experience of the ModTerrain team and the notion of abstracting entire simulation components should be extended toward the ultimate goal of modular simulations that the next section describes.

C. WEB BASED SIMULATION

The Loosely Coupled Components Research Group at the Naval Postgraduate School has developed Java components to support operations research in future distributed military networks. The Java implementations of ModTerrain described in this thesis are examples of loosely coupled components. The research group's Web Based Simulation project seeks to create a library of re-usable Java simulation components for distance learning and to support further research. The ModTerrain API and Java implementations can play a significant role as existing terrain components that can be used off the shelf or modified to meet the needs of the group.

LIST OF REFERENCES

1. Shaara, M., The Killer Angels, Ballantine Books, New York, New York, 1974.
2. Olson, Warren K. Military Operation Research Analyst's Handbook Volume 1, The Military Operations Research Society, Alexandria, Virginia, 1994.
3. Jackson, L., A., and Shirley M. Pratt, ModTerrain API Functional Description, TRADOC Analysis Center, Monterey, California, 1999.
4. Army Model and Simulation Office Home Page: http://www.amso.army.mil/domains/index.htm
5. Combat Training Center Program Handbook, Center For Army Lessons Learned, Fort Monroe, Virginia, 1998.
6. McGlynn, L. and Don Timian, In Pursuit of M&S Standards, Army Model and Simulation Office, Alexandria, Virginia, 1997
7. Army Model and Simulation Office Home Page: http://www.amso.army.mil/sp-div/sc.htm
8. Army Model and Simulation Resource Repository Home Page: http://www.msrr.army.mil/snap/
9. Synthetic Environment Data Representation and Interchange Specification Overview, U.S. Army Simulation, Training and Instrumentation Command (STRICOM), Orlando, Florida, 1998.
10. MILSPEC MIL-V-89039, Vector Smart Map (VMAP Level 0): http://www.nima.mil/publications/specs/printed/VMAP0/vmap0.html
11. U.S. Army Modeling & Simulation Standard Algorithms for Terrain & Dynamic Environment, TRADOC Analysis Center, Fort Leavenworth, Kansas, 1997.
12. Champion, D., The Effect of Feature Data on Line-of-Sight, U.S. Army TRADOC Analysis Center, White Sands Missile Range, New Mexico, 1998.
13. National Image and Mapping Agency Home Page: http://www.nima.mil/geospatial/products/DTED/dted.html
14. Hartman, Parry, Caldwell, High Resolution Combat Modeling, Naval PostGraduate School, Monterey, California, 1992.
15. Champion, D., The Effect of Different Line of Sight Algorithms and Terrain Elevation Representations on Combat Simulations, U.S. Army TRADOC Analysis Center, White Sands Missile Range, New Mexico, 1995.
16. Devore, J. L., Probability and Statistics for Engineering and the Sciences, Fourth Edition, Duxbury Press, Pacific Grove, California, 1995.

APPENDIX A. THE JANUS TERRAIN

This appendix describes in detail the way the JANUS terrain data structure is built. The original code for JANUS was written in FORTRAN. In fact JANUS is a monumental programming achievement in FORTRAN. The entire simulation and tool kit comprises around 350,000 lines of code. An examination of the representation scheme illustrates the close coupling of the terrain model to the simulation. The terrain representation is sparse, and contains only those data needed to support the modeling methodologies used in the simulation. At times this produces an analytical "coarseness" that is remarkably transparent to the user at run time.

Example: Some terrain features contain a set of four REAL variables that indicate the amount of time [in seconds] to achieve an engineering fortification of Level 1, 2, 3, or 4. The notion of four, and only four, distinct levels of a parameter called "fortification" indicates that the underlying simulation methodology is capable of modeling engineer activity that increases survivability. But this model is fairly granular. Replacing the "fortification" model with a more detailed one would require a major change to the terrain representation data structure.

A. JANUS REPRESENTATION SCHEME

JANUS stores terrain data in a large set of arrays. The most important array is the terrain grid. This array does not explicitly contain the x and y coordinates, but rather a one-dimensional array of "grid cell data words" for each post. Because the corner coordinate is held in a variable and the terrain width (both in kilometers and "posts") is also maintained, the post by post x and y coordinates can be calculated efficiently "on the fly." This frees the representation from a large storage requirement. Each grid cell data word is 32 bits that are mapped as shown in Table 5 below. The last 16 bits in the word are used as

Bit	Meaning
0-15	Elevation (pentameters)

Table 5. The JANUS Grid Cell Word

16	Building Present
17	Fence Present
18	Road Present
19	River Present
20	Vegetation Present
21	Urban area Present
22	Generic String Present
23	Generic Area Present
24	Obstacle Present
25	Minefield Present
26	Breach Lane Present
27-31	Not Used

Table 5. The JANUS Grid Cell Word

flags. They indicate only that a terrain feature of the type flagged is present in that grid cell.

B. POLYGONAL FEATURES

Points associated with buildings and other polygonal terrain features are kept in a pool of nodes. The integer variable KNUMTRNNODES specifies the number of nodes in this pool. There are two REAL arrays of length KNUMTRNNODES (TRNODSX and TRNODSY). These contain the X and Y Coordinates of the nodes. This node pool is a complete list of the x and y coordinate of every corner of every polygonal feature in the terrain representation. The number of buildings in the representation is stored in an integer, along with the number of the other supported features. For each polygonal type there is a set of descriptive arrays that store data about the individual features. For buildings these arrays are shown in Table 6.

Primitive Type	Meaning
BYTE	Specifies the building "type"
INT*4	pointer to the node pool for the building outline
BYTE	Specifies the building "type"
INT*4	pointer to the node pool for the building outline
BYTE	number of nodes in pool for this building outline
INT*4	pointer for building's firing ports
BYTE	number of nodes for firing ports
REAL	These are the min & max x & y coordinates for the building
REAL	
REAL	
REAL	

Table 6. The Building Feature Type

Note that this scheme places a 255 vertex limit on individual buildings. This limit is common to all of the polygonal features supported by the JANUS terrain. JANUS terrain representations store an INTEGER that gives the number of poly-features in the file and an array that has pointers to first and last feature of each type.

BYTE	Main type of terrain feature
	1 = Not Used
	2 = Fence
	3 = Road
	4 = River
	5 = Vegetation
	6 = Urban Area
	7 = Generic String
	8 = Generic Area

Table 7. Main Terrain Feature Types

Primitive Type	Meaning
BYTE	Sub-Type of terrain feature
INTEGER*4	Pointer into the node pool
INTEGER*2	Number of nodes for this feature
REAL	These are the min and max x and y coordinates for this feature
REAL	
REAL	
REAL	

Table 8. General Polygonal Feature Data

C. FEATURE SUB-TYPES

There are also a complete set of arrays that permit the inclusion of feature sub-types. Certain sub-types carry additional data elements pertaining to breach lanes, fortification, extinction of LOS, and impact on mobility. Building sub-types are specified in an array sized to the number of sub-types. This array caries the data shown in Table 9.

Primitive Type	Meaning
REAL	Total Height (meters)
REAL	Fractional Area of exterior wall openings
REAL(4)	Engineer minutes to reach fortification levels (1-4)
INTEGER*2	Number of Rooms
BYTE	Construction Type (there can be 255 types)
BYTE	Number of floors
BYTE	Functional Classification

Table 9. Building Sub Types

Fence sub-types are similarly specified with the array data shown in Table 10.

Primitive Type	Meaning
CHARACTER*16	ASCII name
REAL	Total Height (meters)

Table 10. Fence Sub Types

Primitive Type	Meaning
REAL	Oblique Angle (radians) (Maybe oblique angle)
REAL	PLOS at right angle

Table 10. Fence Sub Types

These arrays are further indexed by the number of breach lanes

Primitive Type	Meaning
BYTE	For each fence type
	0 = cross fence
	1 = clear fence
INTEGER*2	For each fence type
	time(minutes) to cross or clear fence

Table 11. Breach Lanes in a Fence Type

Road sub-types are specified in an array sized to the number of supported sub-types by the information in Table 12.

Primitive Type	Meaning
CHARACTER*16	ASCII name
BYTE	1 = Primary, 2 = Secondary
REAL	Road half-width (Kilometers)

Table 12. Road Sub Types

This array is also indexed by the three supported mover types with given speed degradation factors.

Primitive Type	Meaning
BYTE	Speed degradation factor by mover type
	1 = wheeled
	2 = tracked
	3 = footed

Table 13. Road Effect on Mover Types

River sub-types are specified in an array sized by the number of supported river sub-types by

Primitive Type	Meaning
CHARACTER*16	ASCII name
BYTE	1 = primary
	2 = secondary
	3 = filled (lake)
REAL	River half-width (kilometers)

Table 14. River Sub Types

As with roads, river sub-types are indexed by mover type

Primitive Type	Meaning
INTEGER*2	Crossing times by
	1 = wheeled
	2 = tracked
	3 = footed
	4 = swimmer

Table 15. River Effect on Mover Types

Vegetation sub-types are specified in an array sized to the number of sub-types by

Primitive Type	Meaning
CHARACTER*16	ASCII name
BYTE	Height (meters)
REAL	PLOS per 25 meters

Table 16. Data for vegetation sub types

This array is also indexed by mover type

Primitive Type	Meaning
BYTE	Speed degradation factors for

Table 17. Vegetation Effect on Mover Type

Primitive Type	Meaning
	1 = wheeled
	2 = tracked
	3 = footed

Table 17. Vegetation Effect on Mover Type

Urban area sub-types are specified in an array sized to the number of sub-types by

Primitive Type	Meaning
CHARACTER*16	ASCII name
BYTE	Height (meters)
REAL	PLOS per 25 meters

Table 18. Urban Area Sub Types

This array is also indexed by mover type

Primitive Type	Meaning
BYTE	Speed degradation factors for
	1 = wheeled
	2 = tracked
	3 = footed

Table 19. Urban Area Effect on Mover Type

The JANUS terrain also supports features of type generic string and generic area. These have similar data structures to the supported feature types and sub-types, and permit a degree of flexibility in terrain representation by allowing the inclusion of generic features not supported by the general enumeration.

APPENDIX B. MODTERRAIN SOURCE CODE

For the sake of brevity the JAVA source code for the reference implementation of ModTerrain is not provided in this document. It is maintained by the Loosely Coupled Components Group at the Department of Operations Research, Naval Postgraduate School. The source code is licensed under the terms of the GNU General Public Licence.

Copyright (C) 1999 Dale L. Henderson
This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2
of the License, or (at your option) any later version.
This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
See the GNU General Public License for more details.
You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307,
USA.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Rd.,STE 0944
   Fort Belvoir, Virginia 22060-6218

2. ODCSOPS, ATTN: DAMO-ZS 2
   400 Army Pentagon
   Washington, DC 20310-0400

3. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, California 93943-5101

4. Professor Arnold H. Buss, Code OR/Bu. 2
   Department of Operations Research
   Naval Postgraduate School
   Monterey, California 93943-5000

5. Major Leroy Jackson 2
   PO Box 8692
   Monterey, California 93943

6. TRADOC Analysis Center 1
   ATTN: ATRC-TD
   Fort Leavenworth, Kansas 66027-2345

7. TRADOC Analysis Center Research Activities 1
   PO Box 8692
   Monterey, California 93943-0692

8. Major Simon R. Goerger. 1
   TRAC - WSMR, WEC - Combat XXI,
   White Sands Missile Range, NM 88002

9. Captain Dale L. Henderson. 2
   2505 Transit Place
   Anaheim, California 92804